DS5 : Problème du voyageur de commerce

Documents et calculatrice interdits. Le sujet comporte 6 pages et est à traiter en 4h. Il est composé d'un exercice et d'un problème (dont les parties sont largement indépendantes) indépendants.

- Pour un devoir type MPI, traiter les parties du problème dans l'ordre [1-2-3], [5-6-7], 4.
- Pour un devoir type MPI*, traiter les parties du problème dans l'ordre 4, [5-6-7], [1-2-3].

Indiquer sur la copie le type de devoir choisi.

Exercice *Décidable ?*

Pour chacun des problèmes suivants, dire en justifiant la réponse s'il est décidable :

1. ARRET : $\begin{cases}\textbf{Entrée} : \text{Une fonction } f \text{ et un argument } x.\\ \textbf{Question} : \text{Le calcul de } f(x) \text{ termine-t-il ?}\end{cases}$
2. coARRET : $\begin{cases}\textbf{Entrée} : \text{Une fonction } f \text{ et un argument } x.\\ \textbf{Question} : \text{Le calcul de } f(x) \text{ est-il infini ?}\end{cases}$
3. $\text{ARRET}_{\geq 10}$: $\begin{cases}\textbf{Entrée} : \text{Une fonction } f.\\ \textbf{Question} : \text{Le nombre d'arguments } x \text{ pour lesquels } f(x) \text{ termine est-il supérieur à 10 ?}\end{cases}$

Problème *Autour du voyageur de commerce*

Si G est un graphe non orienté, un *cycle hamiltonien* de G est un cycle passant une et une seule fois par chaque sommet de G. Un cycle hamiltonien dans un graphe à n sommets sera noté $(s_0, ..., s_{n-1})$ plutôt que $(s_0, ..., s_{n-1}, s_0)$ (sans répétition du premier sommet).

On s'intéresse au problème du voyageur de commerce, qu'on note PVC. Il s'agit du problème d'optimisation défini comme suit :

$$\begin{cases}\textbf{Entrée} : \text{Un graphe } G = (S, A) \text{ à } n \text{ sommets non orienté, complet et pondéré par } f.\\ \textbf{Solution} : \text{Un cycle hamiltonien } c = (s_0, ..., s_{n-1}) \text{ dans } G.\\ \textbf{Optimisation} : \text{Minimiser le poids } f(c) \text{ de } c, \text{ défini par } f(c) = f(s_0, s_{n-1}) + \sum_{i=0}^{n-2} f(s_i, s_{i+1}).\end{cases}$$

Dans l'ensemble de problème, si $G = (S, A)$ est un graphe, on identifiera l'ensemble S à $\{0, 1, ..., |S| - 1\}$. Sauf mention contraire, $G = (S, A, f)$ est un graphe non orienté, pondéré et complet.

Partie 1 *NP-complétude de PVC*

Dans cette partie, on admet que le problème CH ci-dessous est NP-complet :

$$\begin{cases}\textbf{Entrée} : \text{Un graphe } G = (S, A) \text{ non orienté.}\\ \textbf{Question} : \text{Existe-t-il un cycle hamiltonien dans } G \text{ ?}\end{cases}$$

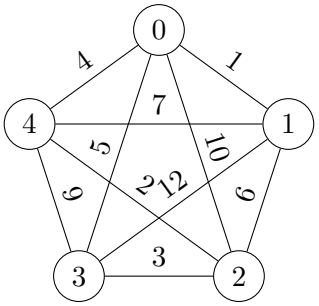
1. Ecrire la version décisionnelle PVC-deci du problème de décision PVC.
2. Montrer que PVC-deci est un problème appartenant à la classe NP.

3. Montrer que PVC-deci est un problème NP-complet.

Partie 2 *Algorithme naïf*

> Les questions de programmation de cette partie et de la partie 3 doivent être écrites en utilisant la syntaxe du langage C. On supposera que les bibliothèques stdlib.h et stdbool.h ont été chargées.

Dans les parties 2 et 3, un graphe non orienté, pondéré et complet sera représenté par sa matrice d'adjacence, laquelle sera implémentée en C par un tableau unidimensionnel d'entiers, les lignes de la matrice étant consécutives dans le tableau. Par exemple, le graphe G_0 en figure 1 sera représenté par le tableau suivant :



$$M_0 = \begin{pmatrix} 0 & 1 & 10 & 5 & 4 \\ 1 & 0 & 6 & 12 & 7 \\ 10 & 6 & 0 & 3 & 2 \\ 5 & 12 & 3 & 0 & 9 \\ 4 & 7 & 2 & 9 & 0 \end{pmatrix}$$

```
int G0[25] = {0, 1, 10, 5, 4,
              1, 0, 6, 12, 7,
              10, 6, 0, 3, 2,
              5, 12, 3, 0, 9,
              4, 7, 2, 9, 0};
```

FIGURE 1 - Le graphe G_0, sa matrice d'adjacence et sa représentation en C.

Par ailleurs, un cycle hamiltonien $c = (s_0, ..., s_{n-1})$ dans un graphe G d'ordre n sera représenté par un tableau contenant les sommets du cycle.

4. Si G est un graphe complet à n sommets, combien y a-t-il de cycles hamiltoniens différents dans G (deux cycles hamiltoniens sont différents s'ils ne sont pas constitués des mêmes arêtes) ?

5. Ecrire une fonction `int f(int* G, int n, int s, int t)` prenant en entrée un tableau correspondant à un graphe pondéré $G = (S, A, f)$, un entier $n = |S|$ et deux entiers $s, t \in S$ et renvoyant le poids de l'arête entre s et t.

6. Ecrire une fonction `int poids_cycle(int* G, int*c, int n)` prenant en entrée un graphe pondéré $G = (S, A, f)$, un cycle c de G et un entier n correspondant à $|S|$ et renvoyant le poids $f(c)$ du cycle c. Par exemple, si `c` est le cycle défini par `{0,2,4,3,1}`, `poids_cycle(G0,c,5)` doit renvoyer 34.

On remarque que toute permutation de $[\![0, n-1]\!]$ forme un cycle hamiltonien dans G. Pour déterminer une solution à PVC on propose naïvement de parcourir toutes les permutations dans l'ordre lexicographique, de calculer le poids de chacune et de conserver l'une de celles de poids minimal. On rappelle que la permutation qui suit $(0, 2, 4, 3, 1)$ dans l'ordre lexicographique est $(0, 3, 1, 2, 4)$.

Si $p = (p_0, ..., p_{n-1})$ est une permutation de $[\![0, n-1]\!]$, on définit les indices suivants :

- $j = \max\{i \in [\![0, n-2]\!] \,|\, p_i < p_{i+1}\}$ et $j = -1$ si cet ensemble est vide.
- $k = \max\{i \in [\![j+1, n-1]\!] \,|\, p_j < p_i\}$ et $k = n$ si $j = -1$.

7. En utilisant les indices j et k, décrire en français ou en pseudo-code un algorithme permettant de modifier une permutation p de $[\![0, n-1]\!]$ en place de sorte à obtenir la permutation suivant p dans l'ordre lexicographique. Si p est la dernière permutation selon l'ordre lexicographique, cet algorithme renvoie faux, sinon il renvoie vrai. On suppose dans la suite que cette fonction est implémentée par la fonction `bool permutation_suivante(int* p, int n)`.

8. Ecrire une fonction int* PVC_naif(int* G, int n) prenant en entrée un graphe G et son nombre de sommets et renvoyant un tableau contenant un cycle hamiltonien de G de poids minimal.

9. Déterminer la complexité temporelle de PVC_naif en fonction du nombre de sommets du graphe en entrée. On admettra pour ce faire que permutation_suivante a une complexité amortie en $O(1)$.

Partie 3 *Heuristique du plus proche voisin*

Afin d'obtenir un cycle hamiltonien de poids faible à défaut d'être minimal en un temps raisonnable, on utilise l'heuristique du plus proche voisin. Cette dernière consiste à partir d'un sommet quelconque du graphe G, l'ajouter au cycle C en construction, puis de répéter les opérations suivantes :

- Regarder quel est le dernier sommet s ajouté à C.
- Déterminer l'ensemble V des voisins de s qui n'ont pas encore été ajoutés à C.
- Parmi les éléments de V, déterminer celui le plus proche de s et l'ajouter à C.

10. Déterminer le cycle renvoyé par l'heuristique du plus proche voisin appliquée au graphe G_0 décrit en figure 1 et en choisissant 0 comme sommet initial. Reprendre la question en partant du sommet 3. L'un de ces cycles est-il de poids minimal dans G_0 ?

11. Ecrire une fonction int plus_proche(int* G, bool* vus, int n, int s) prenant en entrée un graphe $G = (S, A, f)$, un tableau vus de booléens dont au moins une case contient false (on ne le vérifiera pas), un entier n correspondant au nombre de sommets de G et un sommet $s \in S$. Elle renvoie un sommet $t \in S$ vérifiant :

    - vus[t] vaut false.
    - $f(s,t) = \min\{f(s,u) \mid \text{vus[u]} = \text{false}\}$.

    Par exemple, si vus vaut {true, false, true, true, false}, l'appel à plus_proche(G0,vus,5,2) renvoie 4.

12. Ecrire une fonction int* PVC_glouton(int* G, int n) prenant en entrée un graphe $G = (S, A, f)$ et un entier n correspondant à $|S|$ et renvoyant un tableau contenant un cycle hamiltonien de G construit selon l'heuristique du plus proche voisin avec 0 comme sommet initial.

13. Déterminer la complexité de PVC_glouton.

14. Montrer que PVC_glouton n'est pas un algorithme d'approximation à facteur constant de PVC.

15. Dessiner un graphe pondéré complet à 5 sommets tel que l'heuristique du plus proche voisin ne renvoie jamais un cycle hamiltonien de poids minimal, quel que soit le sommet de départ.

Partie 4 *Algorithme de Held-Karp*

Les questions de programmation de cette partie et des suivantes doivent être écrites en utilisant la syntaxe du langage Ocaml. On autorise l'utilisation des fonctions usuelles des modules Array et List.

En Ocaml, les graphes complets seront représentés par matrice d'adjacence. Par exemple, le graphe G_0 décrit en figure 1 est représenté par :

```
let g0 = [| [|0; 1; 3; 5; 4|];
            [|1; 0; 6; 12; 7|];
            [|3; 6; 0; 2; 10|];
            [|5; 12; 2; 0; 9|];
            [|4; 7; 10; 9; 0|]|]
```

Par ailleurs, si S est l'ensemble de sommets d'un graphe, on représentera un sous ensemble $S' \subset S$ par un tableau de booléens de taille $|S|$ contenant `true` en case i si le sommet i appartient à S'.

L'algorithme de Held-Karp est un algorithme de programmation dynamique permettant de résoudre le problème du voyageur de commerce. Pour ce faire, il détermine des solutions aux sous-problèmes suivants : "quel est le chemin de poids minimal entre le sommet s_0 et le sommet s_k passant une et une seule fois par les sommets intermédiaires $s_1, ..., s_{k-1}$?". Le problème qui nous intéresse est celui déterminant un chemin de poids minimal du sommet 0 au sommet 0 et passant par les sommets intermédiaires de $\{1, ..., n-1\}$.

Si $s \in S$ et $S' \subset S \setminus \{0, s\}$ est un ensemble de sommets, on note $P_{\min}(s, S')$ le poids minimal d'un chemin de 0 à s passant une et une seule fois par chaque sommet de S'. On note également $\text{pred}(s, S')$ le sommet qui précède s sur un tel chemin. Par convention, $\text{pred}(0, \emptyset) = 0$.

16. Donner une formule de récurrence vérifiée par $P_{\min}(s, S')$. Expliquer comment calculer $\text{pred}(s, S')$.

Afin d'éviter d'effectuer plusieurs fois le même calcul d'un même $P_{\min}(s, S')$, on choisit de mémoïser les résultats déjà obtenus dans une table de hachage.

17. Expliquer pourquoi il ne faut pas utiliser d'objet mutable comme clé dans une table de hachage.

18. Ecrire une fonction `encodage : int -> bool array -> int*int` prenant en arguments un entier s et un tableau de booléens représentant un sous ensemble $S' \subset S$ et renvoyant un couple d'entiers représentant de manière unique le couple (s, S') en justifiant de cette unicité. Un tel couple d'objets non mutables peut à présent servir de clé dans une table de hachage.

On rappelle que les tables de hachage peuvent être manipulées en Ocaml avec les fonctions suivantes :

- `Hashtbl.create : int -> ('a,'b) Hashtbl.t` prend en entrée un entier m et créé une table de hachage vide occupant un espace mémoire de taille proportionnelle à m (en pratique, on peut prendre $m = 1$).
- `Hashtbl.add : ('a,'b) Hashtbl.t -> 'a -> 'b -> unit` prend en argument une table, une clé et une valeur et ajoute une association dans la table entre cette clé et cette valeur.
- `Hashtbl.mem : ('a,'b) Hashtbl.t -> 'a -> bool` teste si une table contient une clé donnée.
- `Hashtbl.find : ('a,'b) Hashtbl.t -> 'a -> 'b` prend en entrée une table et une clé et renvoie la valeur associée à la clé dans la table si elle existe ; l'exception `Not_found` est levée sinon.

On suppose par ailleurs créées les deux variables globales suivantes :

- Une matrice d'entiers `g` correspondant à un graphe $G = (S, A, f)$.
- Une table de hachage, créée via `let tabh = Hashtbl.create 1`.

19. Ecrire une fonction `chemin_min : int -> bool array -> int*int` qui prend en entrée un sommet $s \in S$ et un sous ensemble $S' \subset S$ du graphe G représenté par `g` et qui calcule en mémoïsant les résultats intermédiaires à l'aide de `tabh` le couple $(P_{\min}(s, S'), \text{pred}(s, S'))$. On expliquera son fonctionnement.

20. En déduire une fonction `pvc_dynamique : unit -> int array` qui renvoie un tableau correspondant à un cycle hamiltonien de poids minimal dans le graphe G représenté par `g`, calculé selon l'algorithme de Held-Karp.

21. Déterminer les complexités spatiale et temporelle de `pvc_dynamique`.

Partie 5 *Algorithme de Prim*

Dans cette partie, $G = (S, A, f)$ est un graphe non orienté, pondéré et connexe mais pas forcément complet. On s'intéresse à l'algorithme de Prim dont le pseudo-code est le suivant :

$B \leftarrow \emptyset$
$R \leftarrow \{s_0\}$ avec s_0 pris au hasard dans S
Tant que $R \neq S$
    $(s,t) \leftarrow$ arête de A de poids minimal telle que $s \in R$ et $t \notin R$
    Ajouter t à R
    Ajouter (s,t) à B
Renvoyer (S,B).

22. Déterminer le graphe renvoyé par l'algorithme de Prim sur le graphe G_1 lorsque $s_0 = 0$.

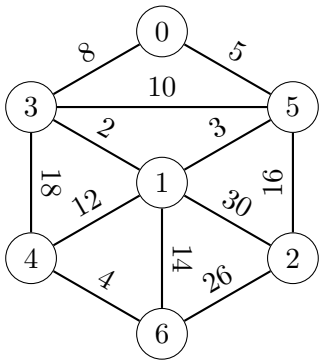


FIGURE 2 - Le graphe G_1.

23. Déterminer la complexité temporelle de l'algorithme de Prim en fonction de $|S|$ et $|A|$. On justifiera en détaillant le choix des structures et les détails d'implémentation jugés pertinents.

24. Montrer que l'algorithme de Prim renvoie un arbre couvrant de G.

On cherche à présent à montrer que l'arbre $T = (S,B)$ renvoyé par l'algorithme de Prim est en fait un arbre couvrant minimal de G. Supposons par l'absurde que ce n'est pas le cas. Alors, si $T' = (S,B')$ est un arbre couvrant minimal de G et $(a_1, ..., a_{n-1})$ les arêtes ajoutées à B par l'algorithme de Prim dans l'ordre d'ajout, comme $T \neq T'$, il existe $i \in [\![1, n-1]\!]$ minimal tel que $a_i \notin B'$. Considérons $T^\star = (S, B^\star)$ un arbre couvrant minimal qui maximise cette valeur de i. On note R l'ensemble des sommets juste avant l'ajout de a_i à B au cours de l'exécution de Prim construisant T.

25. Montrer qu'il existe une arête $a \in B^\star$, reliant un sommet de R et un sommet de $S \setminus R$, telle que $f(a) \geq f(a_i)$. En déduire une contradiction et conclure.

Un graphe non orienté pondéré sera représenté en Ocaml par un tableau de listes d'adjacence :

```
type graphe = (int * int) list array
```

La liste d'adjacence du sommet s contiendra les couples (t,p) tels que t est voisin de s et le poids de l'arête entre s et t vaut p. Par exemple, le graphe G_1 décrit en figure 2 sera représenté par :

```
let g1 = [| [(3, 8); (5, 5)];
            [(2, 30); (3, 2); (4, 12); (5, 3); (6, 14)];
            [(1, 30); (5, 16); (6, 26)];
            [(0, 8); (1, 2); (4, 18); (5, 10)];
            [(1, 12); (3, 18); (6, 4)];
            [(0, 5); (1, 3); (2, 16); (3, 10)];
            [(1, 14); (2, 30); (4, 4)]|]
```

On suppose par ailleurs disposer d'une structure de file de prioirité de type ('a,'b) file_prio dotée des opérations suivantes :

- creer_FP : unit -> ('a,'b) file_prio créé une file de priorité vide.
- ajouter_FP : ('a,'b) file_prio -> 'a -> 'b -> unit prend en entrée une file de priorité, un élément e et une priorité p et ajoute e dans la file avec la priorité p.
- extraire_FP : ('a,'b) file_prio -> 'a modifie la file en entrée de sorte à en supprimer l'élément de priorité minimale et renvoie la valeur de ce dernier.

26. Ecrire une fonction prim : graphe -> graphe prenant en entrée un graphe G non orienté pondéré et connexe (sans le vérifier) et renvoie un graphe correspondant à un arbre couvrant minimal de G calculé à l'aide de l'algorithme de Prim.

Partie 6 *Approximation dans le cadre métrique*

Dans cette partie, $G = (S, A, f)$ est un graphe non orienté, pondéré, complet et tel que la fonction de poids f vérifié l'inégalité triangulaire ; c'est-à-dire :

$$\forall s, t, u \in S,\ f(s, u) \leq f(s, t) + f(t, u)$$

On étend de manière naturelle la fonction f aux sous-graphes et aux chemins de G ; par exemple, le poids d'un sous-graphe est la somme des poids des arêtes qui le composent. On note $c^\star$ un cycle hamiltonien de poids minimal de G et $T = (S, B)$ un arbre couvrant minimal de G.

27. Montrer que pour toute arête $a \in A$, on a $f(a) \geq 0$.
28. Montrer que $f(T) \leq f(c^\star)$.
29. En considérant un parcours en profondeur de T, en déduire l'existence d'un cycle hamiltonien c_T dans G, calculable en temps polynomial et tel que $f(c_T) \leq 2f(c^\star)$.
30. Ecrire une fonction pvc_approx : graphe -> int array de complexité polynomiale et qui soit une 2-approximation de PVC dans le cas où la fonction de pondération vérifie l'inégalité triangulaire. Elle prendra en entrée un graphe $G = (S, A, f)$ représenté par listes d'adjacence et renverra un tableau représentant un cycle hamiltonien de G.

Partie 7 *Non approximabilité dans le cas général*

Dans cette partie, on cherche à montrer que l'hypothèse sur la fonction de poids faite en partie 6 est critique pour permettre de concevoir un algorithme d'approximation du problème du voyageur de commerce.

Supposons qu'il existe une α-approximation (avec $\alpha > 1$) $\mathcal{A}$ de PVC ayant une complexité polynomiale en son entrée $G = (S, A, f)$. Si $G = (S, A)$ est un graphe non orienté avec $|S| > 2$, on considère le graphe non orienté pondéré et complet $K_G = (S, S^2, f)$ suivant :

- Pour $a \in A$, $f(a) = 1$.
- Pour $a \notin A$, $f(a) = \alpha|S|$.

31. Montrer que $G = (S, A)$ possède un cycle hamiltonien si et seulement si K_G possède un cycle hamiltonien de poids inférieur ou égal à $\alpha|S|$.
32. En déduire que sous ces hypothèses, on peut résoudre CH en temps polynomial et conclure.